AD-A103 795 MINNESOTA UNIV MINNEAPOLIS DEPT OF COMPUTER SCIENCE F/G 9/2
PREEMPTIVE SCHEDULING OF A MULTIPROCESSOR SYSTEM WITH MEMORIES --ETC(U)
JUN 81 T H LAI, S SAHNI N00014-80-C-0650
UNCLASSIFIED TR-81-20 NL

1 OF 1
AD A
103795

END
DATE
FILMED
10-81
DTIC

Computer Science Department
136 Lind Hall
Institute of Technology
University of Minnesota
Minneapolis, Minnesota 55455

(9) Technical rept,

/

(14) TR-81-20

(15) N00014-80-C-0650, NSF-MCS80-005856

(6)

Preemptive Scheduling of a Multiprocessor System with Memories To Minimize L_{max}

by

(10) Ten Hwang Lai ~~and~~ Sartaj Sahni

Technical Report 81-20

(11) June 1981

(12) 56

Cover design courtesy of Ruth and Jay Leavitt.

412026

Preemptive Scheduling Of A Multiprocessor System With Memories To Minimize L_{max}[*]

Ten Hwang Lai and Sartaj Sahni
University of Minnesota

Abstract

We develop an $O(k^2n + n\log n)$ algorithm to obtain a preemptive schedule that minimizes L_{max} when n jobs with given memory requirements are to be scheduled on m processors $(n \geq m)$ of given memory sizes. k is the number of distinct due dates. The value of the minimum L_{max} can itself be found in $O(kn + n\log n)$ time.

Key Words and Phrases

Preemptive scheduling, L_{max}, memory requirements.

*This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650 and in part by the National Science Foundation under grant MCS80-005856.

1. Introduction

The problem of scheduling n jobs on a multiprocessor system consisting of m processors, each having its own independent memory of size J_i has been considered by Kafura and Shen [3]. Associated with each job is a processing time t_i and a memory requirement m_i. Job j can be processed on processor i iff $m_j \leq J_i$. No job can be simultaneously processed on two different processors and no processor can process more than one job at any given time instance. In a preemptive schedule, it is possible to interrupt the processing of a job and resume it later on a possibly different processor. In a nonpreemptive schedule, each job is processed without interruption on a single processor.

Obtaining minimum finish time nonpreemptive schedules is NP-hard even when $m = 2$ and $J_1 = J_2$ [2]. Hence, Kafura and Shen [3] study the effectiveness of several heuristics for nonpreemptive scheduling. For the preemptive case, they develop an O(nlogn) algorithm that obtains minimum finish time schedules (without loss of generality, we may assume $n \geq m$). Their algorithm begins by first computing the finish time, f^*, of a minimum finish time schedule. This is done as follows. First, the jobs and processors are reordered such that $J_1 \geq J_2 \geq \cdots \geq J_m$ and $m_1 \geq m_2 \geq \cdots \geq m_n$. This reordering takes O(nlogn) time (again, we assume $n \geq m$). Let F_i be the set of all jobs that can be processed only on processors 1, 2, ..., i because of their memory requirements. Let X_i be the sum of the processing requirements of the jobs in F_i. $X_i = 0$ iff $F_i \doteq \phi$. Kufura and Shen [3] show that

$$f^* = \max\{ \max_i\{t_i\}, \max_i\{X_i/i\}\}. \qquad (1.1)$$

The jobs may now be scheduled in the above order ($m_1 \geq m_2 \geq \cdots \geq m_n$) using f^* and McNaughton's rule [4].

In this paper, we extend the work of [3] to the case when each job has a due time d_i associated with it. Every job is released at a common release time $c_\emptyset$. We are interested in first determining whether or not the n jobs can be preemptively scheduled in such a way that every job completes by its due time. A schedule that has this property is called a <u>feasible schedule</u>.

In Section 2, we show that the existence of a feasible schedule can be determined in polynomial time using network flow techniques. The complexity of the algorithm that results from this approach is $O(kn(n+kr)\log^2(n+kr))$ where k is the number of distinct due dates and r the number of different memory sizes in $\{J_1, J_2, \cdots, J_m\}$. In fact, a feasible schedule (whenever one exists) may be obtained in this much time. In Section 3, we develop another algorithm for this problem. This algorithm is considerably harder to prove correct but has a complexity that is only $O(kn + n\log n)$. A feasible schedule can be constructed in $O(k^2n + n\log n)$ time. In arriving at the algorithm of Section 3, we develop a necessary and sufficient condition for the existence of a feasible schedule. With the help of this condition, in Section 4, we develop an algorithm to obtain a schedule that minimizes the maximum lateness. This algorithm is also of complexity $O(k^2n + n\log n)$.

Sahni [5] and Sahni and Cho [6 and 7] have done work related to that reported here. They have considered preemptive scheduling of n jobs with due dates when $J_1 = J_2 = \cdots = J_m$. For the special case when all memory sizes are the same, Sahni [5] has developed an $O(n\log mn)$ algorithm to obtain a feasible schedule (when one exists). Sahni and Cho [6 and 7] have obtained efficient algorithms for the case when $J_1 = J_2 = \cdots = J_m$ and the processors run at different speeds.

2. The Network Flow Algorithm

In this section, we develop a conceptually simple algorithm to determine a feasible schedule whenever such a schedule exists. This algorithm consists of two phases. Let $c_1, c_2, \cdots, c_k$ be the distinct due times in the multiset $\{d_1, d_2, \ldots, d_n\}$. We may assume that $c_\emptyset < c_1 < \cdots < c_k$ (recall that $c_\emptyset$ is the common release time). In the first phase of our algorithm, we determine the amount $\delta_{i,j}$ of job i that is to be processed in the interval c_{j-1} to c_j. This is done using network flows. In addition to the source (s) and sink (t) nodes, the network to be constructed consists of job nodes, interval nodes, and summation nodes.

For each job there is a job node. The source node connects to each job node via a directed edge. The capacity of the edge to job node i is t_i. If job i has a due time c_j, then there are exactly j interval nodes corresponding to it. These nodes represent the time intervals $[c_\emptyset,c_1]$, $[c_1,c_2]$, ..., $[c_{j-1},c_j]$. From each job node there is a directed edge to each of its interval nodes. Each such edge has a capacity equal to the length of the interval represented by the interval node. In Figure 2.1, the interval node labeled (i,j) corresponds to job i and time interval $[c_{j-1},c_j]$.

The summation nodes are divided out into k summation chains with each chain being used to sum up the flows through each of the k sets of interval nodes. Let $Q_1, Q_2, \cdots, Q_r$ $(Q_1 > Q_2 > \cdots > Q_r)$ be the distinct memory sizes in the multiset $\{J_1, J_2, \cdots, J_m\}$. Let z_i be the number of processors of size Q_i, and let $p_i = \sum_{j=1}^{i} z_j$. Let $Q_{r+1} = \emptyset$. There are exactly r summation nodes in each summation chain. The uth node in chain j is labeled (u,j) in Figure 2.1. There is an edge with capacity c_j-c_{j-1} from the interval node (i,j) to the summation node (u,j) iff

$Q_{u+1} < m_i \leq Q_u$. From node u of summation chain j to node u+1 there is an edge with capacity $p_u(c_j - c_{j-1})$, $1 \leq u < r$, $1 \leq j \leq k$. The last node of each chain j connects to the sink node and has capacity $m(c_j - c_{j-1})$.

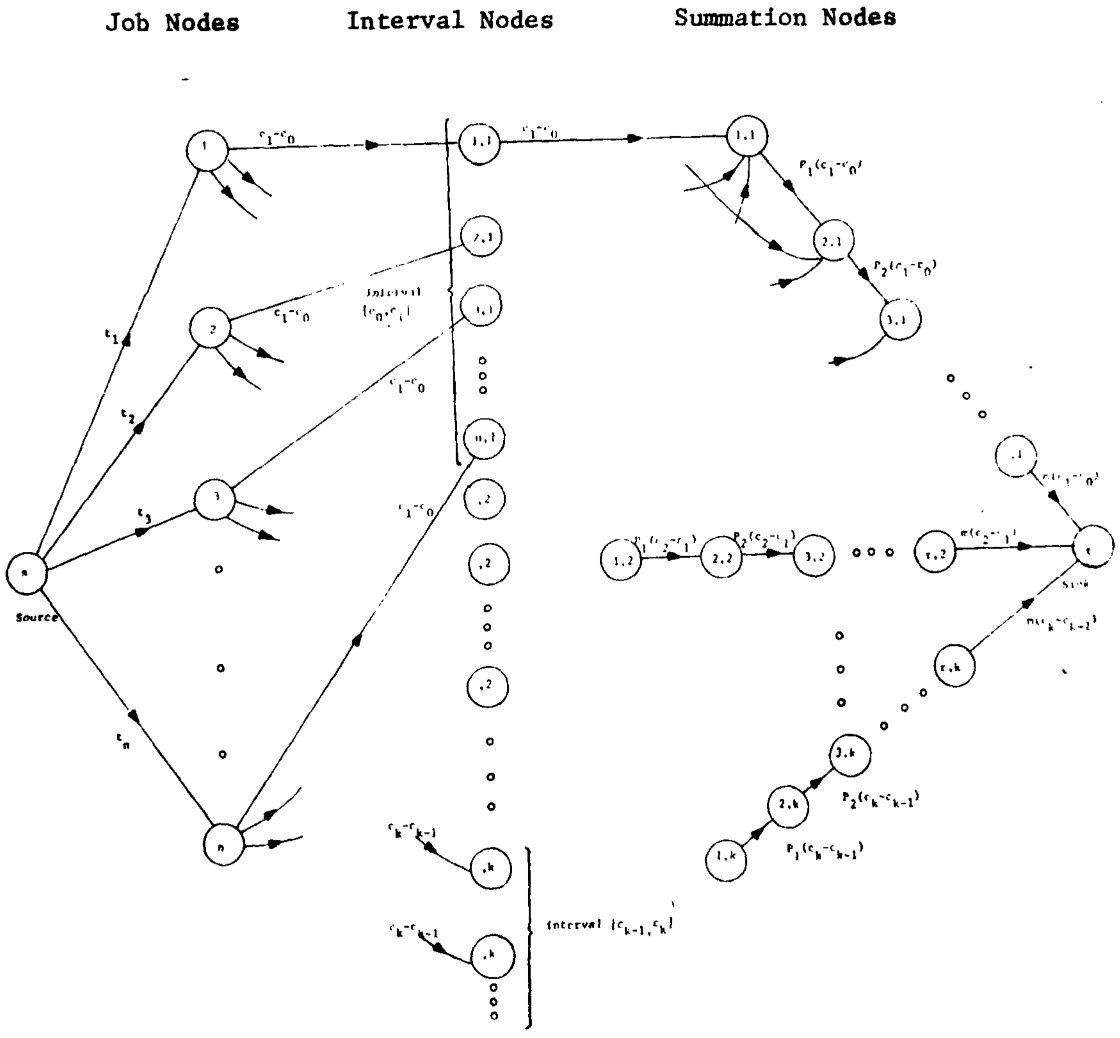

Figure 2.1

We claim that there is a feasible schedule for the n jobs iff the maximum flow through the constructed network is $\sum_{i=1}^{n} t_i$. To see this, first observe that the flow cannot exceed $\sum_{i=1}^{n} t_i$. Suppose that there is a flow of $\sum_{i=1}^{n} t_i$ in the network. Let $\delta_{i,j}$ be the flow in the edge from job node i to interval node (i,j). From the capacity on the edges from job nodes to interval nodes, it is clear that $\delta_{i,j} \le c_j - c_{j-1}$. From this, the capacities on the edges from summation nodes, and Eq.(1.1), it is clear that $\delta_{i,j}$, $1 \le i \le n$ can be scheduled in the interval $[c_{j-1}, c_j]$ using the Kafura-Shen algorithm, $1 \le j \le k$. Hence, there is a feasible schedule when the maximum flow equals $\sum_{i=1}^{n} t_i$. It is readily seen that the reverse is also true; i.e., when there is a feasible schedule there is a flow of value $\sum_{i=1}^{n} t_i$.

So, in phase 1 of our algorithm we construct the flow network and determine the maximum flow. If this is less than $\sum_{i=1}^{n} t_i$, then no feasible schedule exists. If the maximum flow equals $\sum_{i=1}^{n} t_i$, then in phase 2 a feasible schedule is constructed using the Kafura-Shen algorithm together with the interval flows $\delta_{i,j}$ in the maximum flow.

In determining the complexity of this approach, we first note that the interval nodes are easily eliminated. Since exactly one edge enters an interval node and exactly one leaves, these two edges may be combined into one. The total number of nodes in the resulting network is n+kr+2. The number of edges is O(kn). A maximum flow in a v node e edge network can be found in $O(ev\log^2 v)$ time [1]. Hence, the maximum flow in our network can be determined in $O(kn(n+kr)\log^2(n+kr))$ time. Following this, k applications of the Kafura-Shen algorithm are needed. Hence the total time needed to find a feasible schedule (when one exists) is $O(kn(n+kr)\log^2(n+kr))$.

3. An $O(k^2n + n\log n)$ Algorithm

An $O(k^2n + n\log n)$ algorithm (where k is the number of distinct due times) to obtain a feasbile preemptive schedule for a set of n jobs may be arrived at in the following way. As before, each job is characterized by a triple (t_i,d_i,m_i) where t_i is the task time of job i; d_i is its due time; and m_i its memory requirement. Let $c_1, c_2, \ldots, c_k$, $c_1 < c_2 < \cdots < c_k$, denote the distinct due times in the multiset $\{d_1, d_2, \ldots, d_n\}$. Let $c_\emptyset$ be the common release time for the n jobs. Without loss of generality, we may assume that $c_\emptyset < c_1$. Further, we may assume that the jobs are ordered by their memory requirements; i.e., $m_1 \geq m_2 \geq \cdots \geq m_n$.

The m processors are assumed to be ordered by their memory size, i.e., $J_1 \geq J_2 \geq \ldots \geq J_m$. Let $M_\emptyset=\emptyset$, M_1, M_2, $\cdots$, $M_r=m$ be such that processors M_i+1, M_i+2, $\cdots$, M_{i+1} have the same memory size and $J_{M_{i+1}} > J_{M_{i+1}+1}$, $\emptyset \leq i < r$ (for convenience, assume that $J_{m+1}=\emptyset$). The processors $M_i+1, \ldots, M_{i+1}$ define processor class i+1, $\emptyset \leq i<r$. Similary, let $N_\emptyset=\emptyset$, N_1, N_2, $\ldots$, $N_r=n$ be such that jobs N_i+1, N_i+2, $\ldots$, N_{i+1} have a memory requirement that is larger than $J_{M_{i+1}+1}$ but not larger than $J_{M_{i+1}}$, $\emptyset \leq i<r$. It should be clear that a job j such that $N_i < j \leq N_{i+1}$ can be processed only on processors 1, 2, $\ldots$, M_{i+1}. Jobs $N_i+1, \ldots, N_{i+1}$ define job class i+1.

It is easy to see that in every feasible schedule for the n jobs, at least

$$b(i,j) = \begin{cases} t_i & d_i \leq c_j \\ t_i-\min\{t_i,d_i-c_j\} & \text{otherwise} \end{cases}$$

amount of job i must be completed by c_j, $1 \leq i \leq n$, $0 \leq j \leq k$. Observe that if there exist i and j such that $b(i,j) > c_j - c_0$, then there is no feasible schedule.

Of the minimum amount $b(i,j)$ that must be completed before c_j, at most

$$a(i,j) = \min\{b(i,j), c_1 - c_0\}$$

can be completed by c_1.

Define $B(i,j)$ to be the sum of the $b(q,j)$s for those jobs q in job class i. Define $A(i,j)$ in a similar manner. Specifically,

$$B(i,j) = \sum_{q=N_{i-1}+1}^{N_i} b(q,j), \quad 1 \leq i \leq r, \quad 0 \leq j \leq k,$$

and

$$A(i,j) = \sum_{q=N_{i-1}+1}^{N_i} a(q,j), \quad 1 \leq i \leq r, \quad 0 \leq j \leq k.$$

$B(i,j)$ gives the minimum amount of job class i that must be completed by c_j. $A(i,j)$ gives the maximum amount of $B(i,j)$ that can be done by c_1.

One may easily verify that

$$0 \leq b(i,j) \leq b(i,j+1), \quad 1 \leq i \leq n, \quad 0 \leq j < k. \qquad (3.1a)$$

$$0 \leq a(i,j) \leq a(i,j+1), \quad 1 \leq i \leq n, \quad 0 \leq j < k. \qquad (3.1b)$$

$$0 \leq B(i,j) \leq B(i,j+1), \quad 1 \leq i \leq r, \quad 0 \leq j < k. \qquad (3.1c)$$

$$0 \leq A(i,j) \leq A(i,j+1), \quad 1 \leq i \leq r, \quad 0 \leq j < k. \qquad (3.1d)$$

Lemma 1 obtains an interesting inequality between A(i,j) and B(i,j). This inequality will be used later.

__Lemma 1__: If $t_q \leq d_q - c_\emptyset$, $1 \leq q \leq n$, then

$$\frac{B(i,j)-A(i,j)}{c_j-c_1} \leq \frac{B(i,j)}{c_j-c_\emptyset}, \quad 1 \leq i \leq r, \ 1 < j \leq k.$$

__Proof__: Assume that $t_q \leq d_q - c_\emptyset$, $1 \leq q \leq n$. Let q be in the range $[N_{i-1}+1, N_i]$. Since $a(q,j) = \min\{b(q,j),\ c_1-c_\emptyset\}$, $a(q,j) = b(q,j)$ or $a(q,j) = c_1-c_\emptyset$.

If $a(q,j) = b(q,j)$, then

$$\frac{c_1-c_\emptyset}{c_j-c_\emptyset} b(q,j) \leq a(q,j)$$

as $c_1 \leq c_j$.

If $a(q,j) = c_1-c_\emptyset$, then since $t_q \leq d_q-c_\emptyset$ implies $b(q,j) \leq c_j-c_\emptyset$, we get

$$\frac{c_1-c_\emptyset}{c_j-c_\emptyset} b(q,j) \leq c_1-c_\emptyset = a(q,j).$$

So, in both cases we have

$$\frac{c_1-c_\emptyset}{c_j-c_\emptyset} b(q,j) \leq a(q,j).$$

Hence,

$$\frac{c_1-c_\emptyset}{c_j-c_\emptyset} \sum_{q=N_{i-1}+1}^{N_i} b(q,j) \leq \sum_{q=N_{i-1}+1}^{N_i} a(q,j)$$

or

$$\frac{c_1-c_\emptyset}{c_j-c_\emptyset} B(i,j) \leq A(i,j)$$

or

$$\frac{c_1-c_\emptyset}{c_j-c_1}B(i,j) \le \frac{c_j-c_\emptyset}{c_j-c_1}A(i,j)$$

or

$$\frac{c_j-c_\emptyset}{c_j-c_1}[B(i,j)-A(i,j)] \le B(i,j)$$

or

$$\frac{B(i,j)-A(i,j)}{c_j-c_1} \le \frac{B(i,j)}{c_j-c_\emptyset}. \quad []$$

Define a capacity function C(i,j) such that

$$C(i,j) = (M_i-M_{i-1})(c_j-c_\emptyset), \quad 1\le i\le r, \ \emptyset\le j\le k.$$

C(i,j) gives the available processing capacity in processor class i from the release time $c_\emptyset$ to the due time c_j.

Let π be the set of all nonincreasing functions σ with domain $\{\emptyset, 1, 2, \ldots, r\}$ and range $\{\emptyset, 1, \ldots, k\}$. Recall that r is the number of processor classes and k the number of distinct due times. Thus

$$\pi = \{ \sigma \mid \sigma:\{\emptyset,1,\ldots,r\}\to\{\emptyset,1,\ldots,k\} \text{ and } \sigma(i)\ge\sigma(i+1), \emptyset\le i<r\}.$$

π defines the set of <u>profile functions</u>. For example, consider the case r = 4, k = 5, and the profile function σ such that $\sigma(\emptyset) = \sigma(1) = \sigma(2) = 4$; $\sigma(3) = 2$; and $\sigma(4) = 1$. Figure 3.1 displays σ pictorially.

One readily sees that if the given job set has a feasible schedule, then it must be the case that

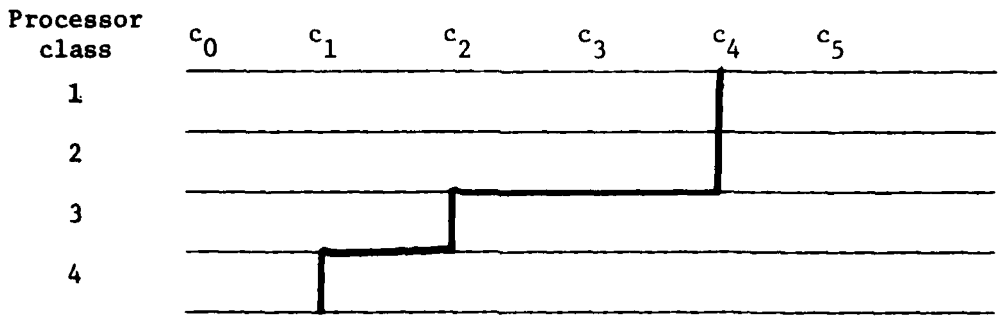

Figure 3.1 Example σ

$$\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i))$$

for all s, $1 \leq s \leq r$ and all $\sigma \prec \pi$. In particular, if there is a feasible schedule, then

$$\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i)) \qquad (3.2)$$

for every $\sigma \prec \pi$. We shall show in Theorem 3.1 that there is a feasible schedule iff (3.2) holds.

We note that for any $\sigma \prec \pi$, if $B(q,\sigma(q)) > C(q,\sigma(q))$ then at least $B(q,\sigma(q)) - C(q,\sigma(q))$ amount of class q jobs must be done on processor classes 1, 2, ..., q-1 by time $c_{\sigma(q)}$. Hence $B(q,\sigma(q)) - C(q,\sigma(q))$ gives a lower bound on the overflow from processor class q to processor classes 1, 2, ..., q-1.

Define π_j as below:

$$\pi_j = \{ \sigma \mid \sigma \prec \pi \text{ and } \sigma(i) \leq j,\ 0 \leq i \leq r\}.$$

We see that for any profile $\sigma \prec \pi_j$,

$$\sum_{q=i+1}^{r} B(q,\sigma(q)) - \sum_{q=i+1}^{r} C(q,\sigma(q))$$

gives a lower bound on the total overflow under this profile from processor classes i+1, ..., r to processor class i, $0 \leq i \leq r$. Hence,

$$Z(i,j) = \max_{\sigma \prec \pi_j} \{ \sum_{q=i+1}^{r} B(q,\sigma(q)) - \sum_{q=i+1}^{r} C(q,\sigma(q))\} \quad (3.3)$$

is also a lower bound on the overflow from processor classes r, r-1 ,..., i+1 to the processor class i up to time c_j.

From (3.3), it follows that $Z(r,j) = 0$, $0 \leq j \leq k$. Also, if $t_i \leq d_i - c_0$, $1 \leq i \leq n$, then $b(i,0) = 0$,$1 \leq i \leq n$ and we obtain

$$Z(i,0) = 0,\ 0 \leq i \leq r. \quad (3.4)$$

Furthermore, since $\sigma:\{0,1,...,r\} \to \{0\}$ is in π_j for all j, it follows from (3.3) that when $t_i \leq d_i - c_0$, $1 \leq i \leq n$, then

$$Z(i,j) \geq 0,\ 0 \leq i \leq r,\ 0 \leq j \leq k. \quad (3.5)$$

From (3.3) we may obtain a simple recurrence for Z(i,j). Let $\sigma' \prec \pi_j$ be the σ at which

$$\sum_{q=i+1}^{r} B(q,\sigma(q)) - \sum_{q=i+1}^{r} C(q,\sigma(q))$$

is maximum. Assume that i < r. If $\sigma'(i+1) \neq j$, then $Z(i,j) = Z(i,j-1)$. If $\sigma'(i+1) = j$, then

$$Z(i,j)=\sum_{q=i+1}^{r} B(q,\sigma'(q))-\sum_{q=i+1}^{r} C(q,\sigma'(q))$$

$$=\sum_{q=i+2}^{r} B(q,\sigma'(q))-\sum_{q=i+2}^{r} C(q,\sigma'(q))+B(i+1,j)-C(i+1,j)$$

$$=Z(i+1,j)+B(i+1,j)-C(i+1,j).$$

This yields:

$$Z(i,j)=\begin{cases} \emptyset & \text{if } i = r \\ Z(i,\emptyset) & \text{if } j = \emptyset \\ \max\{Z(i,j-1),\ Z(i+1,j)+B(i+1,j)-C(i+1,j)\} & \text{otherwise.} \end{cases}$$

(3.6)

Define D(i,j) as below:

$$D(i,j) = \begin{cases} \emptyset & j = \emptyset \\ C(i,1) & \text{otherwise.} \end{cases}$$

Let $\sigma \prec \pi$ be a profile function. $B(q,\sigma(q))-A(q,\sigma(q))$ is a lower bound on the amount of class q job processing that must be done between c_1 and $c_{\sigma(q)}$. Hence, $B(q,\sigma(q))-A(q,\sigma(q))-C(q,\sigma(q))+D(q,\sigma(q))$ (abbreviated as $[B-A-C+D](q,\sigma(q))$) is a lower bound on the overflow from class q jobs from time c_1 to time $c_{\sigma(q)}$. Consequently X(i,j) as defined below:

$$X(i,j) = \max_{\sigma\prec\pi_j} \{ \sum_{q=i+1}^{r} [B-A-C+D](q,\sigma(q))\} \qquad (3.7)$$

is a lower bound on the overflow from processor classes r,...,i+1 to the class i from time c_1 to c_j.

From (3.7), it follows that

$$X(r,j) = 0, \quad 0 \le j \le k. \tag{3.8}$$

Also, if $t_i \le d_i - c_0$, $1 \le i \le n$, then $b(i,0) = a(i,0) = 0$, and $b(i,1) = a(i,1)$, $1 \le i \le n$. From this and (3.7), we obtain

$$X(i,0) = X(i,1) = 0, \quad 0 \le i \le r. \tag{3.9}$$

From (3.7) the recurrence

$$X(i,j) = \begin{cases} 0 & i=r \\ X(i,0) & j=0 \\ \max\{X(i,j-1),[X+B-A-C+D](i+1,j)\} & \text{otherwise} \end{cases} \tag{3.10}$$

may be obtained in the same way as (3.6) was obtained from (3.3).

From (3.10), we see that if $t_i \le d_i - c_0$, $1 \le i \le n$, then

$$X(i,j) \ge 0, \quad 0 \le i \le r, \quad 0 \le j \le k. \tag{3.11}$$

The difference between Z and X will play an important role in the development of our algorithm. Define:

$$Y(i,j) = Z(i,j) - X(i,j), \quad 0 \le i \le r, \quad 0 \le j \le k.$$

We establish in Lemma 3 an important inequality on Y. Before proving this Lemma, we obtain some results needed in its proof.

<u>Lemma 2</u>: If $t_q \le d_q - c_0$, $1 \le q \le n$, then

$$(c_j - c_0)X(i,j) \le (c_j - c_1)Z(i,j), \quad 0 \le i \le r, \quad 0 \le j \le k.$$

<u>Proof</u>: Assume that $t_q \le d_q - c_0$, $1 \le q \le n$. The proof is by induction on i and j.

Induction Base(1) When $i = r$, $X(i,j) = Z(i,j) = 0$.

Induction Hypothesis(1) Assume that the inequality is correct for all j when $0 < i = p \leq r$.

Induction Step(1) When $i = p-1$, the inequality may be shown correct by induction on j.

I.B.(2) When $j = 0$ or 1, $X(p-1,j) = 0$ and $Z(p-1,j) \geq 0$.

I.H.(2) Assume that the inequality is correct when $k>j=q\geq 1$.

I.S.(2) Let $j = q+1$. From Eq. (3.10), we see that there are two possibilities for $X(p-1,q+1)$.

Case(i) $X(p-1,q+1) = X(p-1,q)$: In this case,

$$\begin{aligned}(c_q-c_0)X(p-1,q+1) &= (c_q-c_0)X(p-1,q)\\ &\leq (c_q-c_1)Z(p-1,q) \qquad \text{(I.H.(2))}\\ &\leq (c_q-c_1)Z(p-1,q+1) \qquad \text{(Eq.(3.6))}\end{aligned}$$

Since $c_q - c_0 > c_q - c_1 \geq 0$, we get

$$X(p-1,q+1) \leq Z(p-1,q+1)$$

or

$$(c_{q+1}-c_q)X(p-1,q+1) \leq (c_{q+1}-c_q)Z(p-1,q+1).$$

Adding this inequality to the previous one yeilds:

$$(c_{q+1}-c_0)X(p-1,q+1) \leq (c_{q+1}-c_1)Z(p-1,q+1).$$

Case(ii) $X(p-1,q+1) = [X+B-A-C+D](p,q+1)$: Now, we obtain:

$$\frac{X(p-1,q+1)}{c_{q+1}-c_1} = \frac{[X+B-A-C+D](p,q+1)}{c_{q+1}-c_1}.$$

Using I.H.(1), Lemma 1, and the equality

$$\frac{C(p,q+1)-D(p,q+1)}{c_{q+1}-c_1} = M_p-M_{p-1} = \frac{C(p,q+1)}{c_{q+1}-c_\emptyset},$$

we obtain:

$$\frac{X(p-1,q+1)}{c_{q+1} - c_1} \leq \frac{Z(p,q+1) + B(p,q+1) - C(p,q+1)}{c_{q+1} - c_\emptyset}$$

$$\leq \frac{Z(p-1,q+1)}{c_{q+1} - c_\emptyset} \qquad \text{(Eq.(3.10))} \qquad []$$

Corollary 1: If $t_q \leq d_q-c_\emptyset$, $1 \leq q \leq n$, then

$$Y(i,j) \geq \emptyset, \ \emptyset\leq i\leq r, \ \emptyset\leq j\leq k.$$

Proof: For $j \geq 1$, this follows from Lemma 2 and the fact $c_j-c_\emptyset > c_j-c_1 \geq \emptyset$. For $j = \emptyset$, this follows from Eqs. (3.4) and (3.9). []

Lemma 3: If $t_q \leq d_q - c_\emptyset$, $1 \leq q \leq n$, then

$$Y(i,j) \geq Y(i,j-1), \ \emptyset\leq i\leq r, \ 1\leq j\leq k.$$

Proof: Assume that $t_q \leq d_q - c_\emptyset$, $1 \leq q \leq n$. The proof is by induction on i and j.

I.B.(1) When $i = r$, $Y(i,j) = Y(i,j-1) = \emptyset$, $1\leq j\leq k$.

I.H.(1) Assume that $Y(i,j) \geq Y(i,j-1)$ for all j, when $1 \leq u < i \leq r$ where u is an arbitrary integer in the range [1,r-

1].

<u>I</u>.<u>S</u>.(<u>1</u>) We need to show that $Y(u,j) \geq Y(u,j-1)$, $1 \leq j \leq k$.

<u>I</u>.<u>B</u>.(<u>2</u>) When $j=1$, $Y(u,1) \geq 0$ (Corollary 1) and $Y(u,0)=0$.

<u>I</u>.<u>H</u>.(<u>2</u>) Assume that $Y(u,j) \geq Y(u,j-1)$, $1 \leq j < b$.

<u>I</u>.<u>S</u>.(<u>2</u>) We need to show that $Y(u,b) \geq Y(u,b-1)$.

<u>Case</u> (<u>i</u>) $Z(u,b) \geq Z(u,b-1)$ and $X(u,b) = X(u,b-1)$: In this case, it is readily seen that $Y(u,b) \geq Y(u,b-1)$.

<u>Case</u> (<u>ii</u>) $Z(u,b) = Z(u,b-1)$ and $X(u,b-1) < X(u,b) = [X+B-A-C+D](u+1,b)$:

This case is not possible. To see this, suppose that this case is possible. Let $a \geq u$ be the largest a for which

$$Z(a,j)=Z(a,j-1) \text{ and } X(a,j-1)<[X+B-A-C+D](a+1,j) \quad (3.12)$$

for some j. Let c be the smallest j for which (3.12) holds. Note that $c > 0$. So,

$$Z(a,c)=Z(a,c-1) \text{ and } X(a,c-1)<[X+B-A-C+D](a+1,c). \quad (3.13)$$

Since $X(a,c-1) \geq 0$, it follows from Eqs. (3.10) and (3.13) that $X(a,c) > 0$. Assume that

$$Z(a,c) = Z(a,c-1) = Z(a,c-2) = \cdots = Z(a,v) \neq Z(a,v-1).$$

Then, it follows from our choice of a and c that

$$X(a,c-1) = X(a,c-2) = \cdots = X(a,v).$$

If $v = 0$, then $0 = Z(a,v) = Z(a,c) \geq X(a,c) > 0$. Hence, $v \neq$

Ø. Now, from the choice of v, we get

$$\begin{aligned} Y(a,v) &= Z(a,v) - X(a,v) \\ &\le [Z+B-C](a+1,v) - [X+B-A-C+D](a+1,v) \\ &= [Y+A-D](a+1,v). \end{aligned} \qquad (3.14)$$

Also,

$$X(a,c-1) < [X+B-C-A+D](a+1,c),$$

and

$$Z(a,c-1) \ge [Z+B-C](a+1,c).$$

So,

$$Y(a,v) = Y(a,c-1) > [Y+A-D](a+1,c).$$

Substituting into (3.14) yields:

$$[Y+A-D](a+1,v) > [Y+A-D](a+1,c)$$

or,

$$\begin{aligned} Y(a+1,v) &> [Y+A-D](a+1,c) - [A-D](a+1,v) \\ &\ge Y(a+1,c) \qquad \text{(Eq. (3.1) and definition of D)} \end{aligned}$$

But, $a + 1 > u$ and so from I.H.(1), it follows that

$$Y(a+1,c) \ge Y(a+1,c-1) \ge \cdots \ge Y(a+1,v).$$

So, case (ii) is not possible.

<u>Case</u> (<u>iii</u>) $Z(u,b) = [Z+B-C](u+1,b)$ and $X(u,b) = [X+B-A-C+D](u+1,b)$:

Now, $Y(u,b) = [Y+A-D](u+1,b)$. Suppose that

$$Z(u,b-1) = Z(u,b-2) = \cdots = Z(u,v) \neq Z(u,v-1). \quad (3.15)$$

From the proof of case (ii), it follows that $X(u,b-1) = X(u,b-2) = \ldots = X(u,v)$. So, $Y(u,b-1) = Y(u,v)$. If $v = \emptyset$, then $Y(u,b-1) = Y(u,\emptyset) = \emptyset \leq Y(u,b)$. If $v \neq \emptyset$, then

$$\begin{aligned} Y(u,b-1) &= Y(u,v) \\ &\leq [Y+A-D](u+1,v) && \text{(Eqs. 3.15, 3.6, 3.10)} \\ &\leq Y(u+1,b)+[A-D](u+1,v) && \text{(I.H.(1))} \\ &\leq [Y+A-D](u+1,b) && \text{(Eq. 3.1)} \\ &= Y(u,b). && [\,] \end{aligned}$$

We are now ready to describe our preemptive scheduling algorithm. The jobs will be scheduled in k phases. In phase j we determine the amount of each job i that is to be scheduled from c_{j-1} to c_j. Once this amount has been determined, the actual schedule from c_{j-1} to c_j is constructed using the Kafura-Shen algorithm.

Procedure COMPUTE_W determines the amount w_i of job i that is to be scheduled from $c_\emptyset$ to c_1. In this procedure, R_s denotes the amount of idle time remaining on processor classes 1, 2, ..., s following the scheduling of the w_is corresponding to jobs in job classes 1, 2, ..., s. Lemma 4 obtains some interesting properites of R_s. When actually implementing COMPUTE_W, the subscripts on h, R, and Q may be

```
line   Procedure COMPUTE_W
       //$w_i$ is the amount of job i to be processed from
         $c_\emptyset$ to $c_1$//
 1     $R_\emptyset \leftarrow \emptyset$
 2     for s <- 1 to r do //consider jobs by classes//
 3        $Q_s \leftarrow \{ j \mid A(s,j) + Y(s,j) \le R_{s-1} + C(s,1)\}$
 4        if $Q_s = \emptyset$ then print('infeasible job set')
 5                         stop  endif
 6        $h_s \leftarrow \max\{ j \mid j \in Q_s\}$
 7        case
 8              :(1) $h_s = k$: $w_i \leftarrow a(i,k)$, $N_{s-1} < i \le N_s$
 9              :(2) $h_s<k$, $A(s,h_s+1)+Y(s,h_s) \ge R_{s-1}+C(s,1)$:
10                   set $w_i$ such that
                     $a(i,h_s) \le w_i \le a(i,h_s+1)$, $N_{s-1}<i\le N_s$ and

                     $\sum_{i=N_{s-1}+1}^{N_s} w_i+Y(s,h_s) = R_{s-1}+C(s,1)$

11              :(3) else:  $w_i \leftarrow a(i, h_s + 1)$, $N_{s-1}<i\le N_s$
12        end case
13        $R_s \leftarrow R_{s-1} + C(s,1) - \sum_{N_{s-1}+1}^{N_s} w_i$
14     end for
15     end COMPUTE_W
```

Figure 3.2

omitted. We have kept them in the version given in Figure 3.2 so that we may easily refer to the values of h, R, and Q during different iterations of the for loop. One should also note that in case (2), since $A(s,h_s+1) + Y(s,h_s) \ge R_{s-1} + C(s,1)$ and $A(s,h_s) + Y(s,h_s) \le R_{s-1} + C(s,1)$, there exist w_i, $a(i,h_s) \le w_i \le a(i,h_s+1)$, such that

$$\sum_{N_{s-1}+1}^{N_s} w_i + Y(s,h_s) = R_{s-1} + C(s,1).$$

These w_is are easily determined by first setting all $w_i = a(i,h_s)$, $N_{s-1}<i\leq N_s$ and then incrementing the w_is one by one (up to at most $a(i,h_s+1)$) until the desired equality is satisfied.

Lemma 4: If $\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i))$ for every $\sigma \prec \pi$, then

(1) $Q_s \neq \phi$ and $h_s \geq 1$, $1 \leq s \leq r$.

(2) $R_s \geq Y(s, h_s)$, $1 \leq s \leq r$.

(3) $R_s \leq Y(s,h_s+1)$ if $h_s \neq k$, $1 \leq s \leq r$.

Proof: Assume that

$$\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i)) \qquad (3.16)$$

for every $\sigma \prec \pi$. In particular, using $\sigma(i) = \emptyset$, $1 \leq i \leq r$ in (3.16), we obtain $\sum_{i=1}^{r} B(i,\emptyset) \leq \emptyset$. Since $B(i,\emptyset) \geq \emptyset$, $1 \leq i \leq r$, it follows that $B(i,\emptyset) = \emptyset$, $1 \leq i \leq r$ and so $b(i,\emptyset) = \emptyset$, $t_i \leq d_i - c_\emptyset$, and $b(i,1) \leq c_1 - c_\emptyset$, $1 \leq i \leq n$. Hence, $A(i,1) = B(i,1)$, $1 \leq i \leq r$. From this and Eq. (3.9) we obtain

$$A(i,1)+Y(i,1) = B(i,1)+Z(i,1),\ 1\leq i\leq r. \qquad (3.17)$$

Now we shall show (1), (2), and (3) by induction on s.

I.B. From (3.3) and (3.16), it follows that $Z(\emptyset,1) = \emptyset$. From (3.6), it follows that $Z(1,1) + B(1,1) - C(1,1) \leq Z(\emptyset,1) \leq \emptyset$. Combining this with (3.17), we obtain $A(1,1) + Y(1,1) \leq C(1,1)$. Since $R_\emptyset = \emptyset$, it follows that $1 \prec Q_1$. Hence, $Q_1 \neq \phi$ and $h_1 \geq 1$.

By definition, $R_1 = R_\emptyset + C(1,1) - \sum_{N_\emptyset+1}^{N_1} w_i$. From the algorithm we see that if $h_1 = k$, then $\sum_{N_\emptyset+1}^{N_1} w_i = A(1,h_1)$. But, from line 3, it follows that $Y(1,h_1) \leq R_\emptyset + C(1,1) - A(1,h_1)$. Hence, $R_1 \geq Y(1,h_1)$. If $h_1 < k$ and $A(1,h_1+1) + Y(1,h_1) \geq R_\emptyset + C(1,1)$, then from line 10, we obtain $Y(1,h_1) = R_\emptyset + C(1,1) - \sum_{N_\emptyset+1}^{N_1} w_i$. Hence, $R_1 \geq Y(1,h_1)$. Since $w_i \leq a(i,h_1+1)$ in this case, $\sum w_i \leq A(i,h_1+1)$. Also, from the definition of h_1, it follows that $A(1,h_1+1)+Y(1,h_1+1) > R_\emptyset+C(1,1)$. Hence, $R_1 < Y(1,h_1+1)$.

In the third case of the algorithm, $A(1,h_1+1)+Y(1,h_1) < R_\emptyset+C(1,1)$ and $\sum w_i = A(1,h_1+1)$. Hence, $Y(1,h_1) < R_\emptyset + C(1,1) - \sum w_i$ and $R_1 > Y(1,h_1)$. From the definition of h_1 and w_i, it follows that $R_1 < Y(1,h_1+1)$ (same proof as for previous case).

<u>I.H.</u> Assume that (1), (2), and (3) are true for some arbitrary s, $s = q$, $1 \leq q < r$.

<u>I.S.</u> We shall show that (1), (2), and (3) are true when $s = q+1$.

$$\begin{aligned}
R_q &\geq Y(q,h_q) && \text{(induction hypothesis)}\\
&\geq Y(q,1) && \text{(Lemma 3)}\\
&= Z(q,1) && (X(q,1)=\emptyset \text{ from Eqs. 3.16 and 3.9})\\
&\geq [Z+B-C](q+1,1) && \text{(Eq. 3.6)}\\
&= [Y+A-C](q+1,1). && \text{(Eq. (3.17))}
\end{aligned}$$

Hence, $1 \in Q_{q+1}$ and so $Q_{q+1} \neq \emptyset$ and $h_{q+1} \geq 1$. (2) and (3) can now be proved in the same way as in the induction base. []

Before establishing the correctness of our scheme to compute the w_is, we obtain some relationships concerning the amount of processing t'_i of job i that remains to be done following time c_1. Note that $t'_i = t_i - w_i$, $1 \le i \le n$.

Define $b'(i,j)$, $a'(i,j)$, $B'(i,j)$, and $A'(i,j)$ to be the values obtained for b, a, B, and A when t'_i is used in place of t_i. Let $C'(i,j) = (M_i - M_{i-1})(c_j - c_1)$, $1 \le i \le r$, $1 \le j \le k$, and $W(i) = \sum_{q=N_{i-1}+1}^{N_i} w_q$, $1 \le i \le r$. Lemmas 5 and 6 enable us to establish Theorem 1.

<u>Lemma 5</u>: If the condition of lemma 4 is satisfied then

(1) $B'(i,j) \le B(i,j) - A(i,j)$, $j \le h_i$.

(2) $B'(i,j) = B(i,j) - W(i)$, $j > h_i$.

<u>Proof</u>: It is easy to see that for any job q, $1 \le q \le n$,

$$b'(q,j) = \max\{\emptyset,\ b(q,j) - w_q\}.$$

When $j \le h_i$, $a(q,j) \le a(q,h_i)$ (from Eq. (3.1)). From the algorithm, we see that $w_q \ge a(q,h_i)$ in cases (1) and (2) and $w_q = a(q,h_i+1)$ in case (3). Hence, in all cases, we have $w_q \ge a(q,j)$, $N_{i-1}+1 \le q \le N_i$, $j \le h_i$. For $B'(i,j)$, $j \le h_i$, we now obtain

$$B'(i,j) = \sum_{q=N_{i-1}+1}^{N_i} b'(q,j)$$

$$= \sum \max\{\emptyset,\ b(q,j) - w_q\}$$

$$\leq \Sigma \max\{\emptyset,\ b(q,j)-a(q,j)\}$$

$$= \Sigma(b(q,j)-a(q,j)) \qquad (\text{as } b(q,j)\geq a(q,j)\geq \emptyset)$$

$$= B(i,j) - A(i,j).$$

We now consider the case $j > h_i \neq k$. From cases (2) and (3) of the algorithm, Eq (3.1), and the definition of $a(q,j)$, we see that

$$a(q,h_i) \leq w_q \leq a(q,h_i+1) \leq a(q,j) \leq b(q,j).$$

So,

$$b'(q,j) = \max\{\emptyset,\ b(q,j)-w_q\} = b(q,j)-w_q.$$

Hence,

$$B'(i,j) = B(i,j)-W(i). \qquad []$$

For convenience in proving the next lemma, we define $h_\emptyset = 1$.

__Lemma 6__: If $\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i))$ for every $\sigma \prec \pi$, then the following are true for every s, $\emptyset \leq s \leq r$ and every σ such that $\sigma(s) \geq 1$:

(1) $\sum_{i=1}^{s} B'(i,\sigma(i)) + Z(s,\sigma(s)) - R_s \leq \sum_{i=1}^{s} C'(i,\sigma(i))$,

(2) If $\sigma(s)\leq h_s$, then $\sum_{i=1}^{s} B'(i,\sigma(i))+X(s,\sigma(s)) \leq \sum_{i=1}^{s} C'(i,\sigma(i))$.

__Proof__: Assume that

$$\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i)) \text{ for every } \sigma \prec \pi. \quad (3.18)$$

We shall use induction on s.

<u>I</u>.<u>B</u>. From (3.3), we obtain

$$Z(0,\sigma(0)) = \max_{\sigma' \prec \pi_{\sigma(0)}} \{ \sum_{i=1}^{r} B(i,\sigma'(i)) - \sum_{i=1}^{r} C(i,\sigma'(i)) \} (3.19)$$

Using σ' such that $\sigma'(i) = 0$, $0 \leq i \leq r$ in (3.18) yields $B(i,0) = 0$ and hence $t_q \leq d_q - c_0$, $1 \leq q \leq n$. From (3.5), (3.18) and (3.19), we now get $Z(0,\sigma(0)) = 0$. From Corollary 1 and (3.11), we get $0 = Z(0,\sigma(0)) \geq X(0,\sigma(0)) \geq 0$. Hence, $X(0, \sigma(0)) = 0$.

Clearly, $\sum_{i=1}^{0} B'(i,\sigma(i)) = \sum_{i=1}^{0} C'(i,\sigma(i)) = R_0 = 0$.

Hence, when $s = 0$,

$$\sum_{i=1}^{s} B'(i,\sigma(i)) + Z(s,\sigma(s)) - R_s \leq \sum_{i=1}^{s} C'(i,\sigma(i))$$

and

$$\sum_{i=1}^{s} B'(i,\sigma(i)) + X(s,\sigma(s)) \leq \sum_{i=1}^{s} C'(i,\sigma(i))$$

for every $\sigma \prec \pi$.

<u>I</u>.<u>H</u>. Assume that (1) and (2) are true for $s = t$, where t is in the range $0 \leq t < r$.

<u>I</u>.<u>S</u>. We proceed to establish (1) and (2) when $s = t+1$ by considering the three cases $\sigma(t+1) > h_{t+1}$; $\sigma(t+1) \leq h_{t+1}$ and $\sigma(t) \leq h_t$; and $\sigma(t+1) \leq h_{t+1}$ and $\sigma(t) > h_t$.

<u>Case 1</u>: $\sigma(t+1) > h_{t+1}$: We first obtain the following

$$\sum_{i=1}^{t} B'(i,\sigma(i)) + Z(t,\sigma(t)) - R_t \le \sum_{i=1}^{t} C'(i,\sigma(i)) \qquad \text{(I.H.)}$$

$$R_t - R_{t+1} = W(t+1) - C(t+1,1) \qquad \text{(def. of } R_{t+1}\text{)}$$

$$B'(t+1,\sigma(t+1)) + W(t+1) = B(t+1,\sigma(t+1)) \qquad \text{(Lemma 5)}$$

$$Z(t+1,\sigma(t+1))+B(t+1,\sigma(t+1))-C(t+1,\sigma(t+1))$$

$$\le Z(t,\sigma(t+1)) \qquad \text{(Eq.(3.6) and } \sigma(t+1) \ge 1\text{)}$$

$$\le Z(t,\sigma(t)) \qquad (\sigma(t) \ge \sigma(t+1) \text{ and Eq.(3.6))}$$

Adding these four equalities and inequalities yields:

$$\sum_{i=1}^{t+1} B'(i,\sigma(i)) + Z(t+1,\sigma(t+1)) - R_{t+1} \le \sum_{i=1}^{t+1} C'(i,\sigma(i))$$

<u>Case 2</u>: $\sigma(t+1) \le h_{t+1}$ and $\sigma(t) \le h_t$: From the induction hypothesis, we have

$$\sum_{i=1}^{t} B'(i,\sigma(i))+X(t,\sigma(t)) \le \sum_{i=1}^{t} C'(i,\sigma(i)). \qquad (3.20)$$

From Eq.(3.10) and the fact that $\sigma(t) \ge \sigma(t+1) \ge 1$, we get:

$$X(t,\sigma(t)) \ge X(t,\sigma(t+1)) \ge [X+B-A-C+D](t+1,\sigma(t+1)).$$

Using Lemma 5, this reduces to

$$X(t,\sigma(t)) \ge X(t+1,\sigma(t+1))+B'(t+1,\sigma(t+1))-C'(t+1,\sigma(t+1)).$$

Combining with (3.20) yields:

$$\sum_{i=1}^{t+1} B'(i,\sigma(i))+X(t+1,\sigma(t+1)) \le \sum_{i=1}^{t+1} C'(i,\sigma(i)). \quad (3.21)$$

Since $R_{t+1} \ge Y(t+1,h_{t+1}) \ge Y(t+1,\sigma(t+1))$ (Lemmas 4 and 3) we conclude that $Z(t+1, \sigma(t+1)) - R_{t+1} \le X(t+1,\sigma(t+1))$. Substituting into (3.21) yields

$$\sum_{i=1}^{t+1} B'(i,\sigma(i))+Z(t+1,\sigma(t+1))-R_{t+1} \le \sum_{i=1}^{t+1} C'(i,\sigma(i)).$$

<u>case 3</u>: $\sigma(t+1) \le h_{t+1}$ and $\sigma(t) > h_t$: From the induction hypothesis, we get

$$\sum_{i=1}^{t} B'(i,\sigma(i))+Z(t,\sigma(t)) - R_t \le \sum_{i=1}^{t} C'(i,\sigma(i)). \quad (3.22)$$

Since $h_t \neq k$, we obtain from Lemma 4:

$$R_t \le Y(t,h_t+1) \le Y(t,\sigma(t)). \quad (3.23)$$

From Lemma 5, Eq. (3.10), and the inequality $\sigma(t) \ge \sigma(t+1) \ge 1$, we get

$$[X+B'-C'](t+1,\sigma(t+1)) \le X(t,\sigma(t+1)) \le X(t,\sigma(t)). (3.24)$$

Adding (3.22), (3.23), and (3.24) yields:

$$\sum_{i=1}^{t+1} B'(i,\sigma(i))+X(t+1,\sigma(t+1)) \le \sum_{i=1}^{t+1} C'(i,\sigma(i)).$$

Using the same reasoning as in case 2, we may now conclude the truth of (1) when $s = t+1$. []

<u>Theorem 1</u>: There exists a feasible preemptive schedule for the given n jobs if and only if

$$\sum_{i=1}^{r} B(i,\sigma(i)) \le \sum_{i=1}^{r} C(i,\sigma(i)) \text{ for every } \sigma \prec \pi_k.$$

Proof: We have already pointed out that if a feasible schedule exists, then the above inequality is satisfied for every $\sigma \prec \pi_k$. So, we need only show that when the above inequality is satisfied for every $\sigma \prec \pi_k$, there is a feasible schedule. Assume that

$$\sum_{i=1}^{r} B(i,\sigma(i)) \leq \sum_{i=1}^{r} C(i,\sigma(i)) \text{ for every } \sigma \prec \pi_k. \quad (3.25)$$

From (3.25) it is clear that when k = 1, the t_is and Eq (1.1) yield $f^* \leq c_1 - c_\emptyset$ and so a feasible schedule exists.

For the induction hypothesis, we assume that there exists a feasible schedule when (3.25) is satisfied and k=q for some q, $1 \leq q$. We show that if (3.25) is satisfied when k=q+1, then there is a feasible schedule. From Lemma 4, we see that $Q_s \neq \phi$ for any s. Hence procedure COMPUTE_W successfully computes the w_is. It is clear from COMPUTE_W that $w_i \leq a(i,u) \leq c_1 - c_\emptyset$ where $u = h_s+1$ or q+1 and that $\sum_{i=1}^{N_s} w_i \leq M_s(c_1-c_\emptyset) - R_s \leq M_s(c_1-c_\emptyset)$ for every s. Hence, the w_is satisfy (1.1) (i.e., $f^* \leq c_1 - c_\emptyset$) and may be scheduled from $c_\emptyset$ to c_1 using the Kafura-Shen algorithm.

Now, consider the t'_is. We know that X(r,j) = Z(r,j) = Ø, $\emptyset \leq j \leq q+1$. If $h_r < q+1$, then from Lemma 4, we obtain $\emptyset \leq R_r \leq Y(r,h_r+1) = \emptyset$ or $R_r = \emptyset$. Using this in Lemma 6 yields:

$$\sum_{i=1}^{r} B'(i,\sigma(i)) \leq \sum_{i=1}^{r} C'(i,\sigma(i)) \text{ for every } \sigma \prec \pi_{q+1}$$

$$\text{such that } \sigma(r) \geq 1. \quad (3.26a)$$

If $h_r = q+1$ then $\sigma(r) \leq h_r$ and from Lemma 6 we once again get:

$$\sum_{i=1}^{r} B'(i,\sigma(i)) \leq \sum_{i=1}^{r} C'(i,\sigma(i)) \text{ for every } \sigma \prec \pi_{q+1}$$

$$\text{such that } \sigma(r) \geq 1. \qquad (3.26b)$$

One readily sees that (3.26a) and (3.26b) are equivalent to

$$\sum_{i=1}^{r} B'(i,\sigma(i)+1) \leq \sum_{i=1}^{r} C'(i,\sigma(i)+1) \text{ for every } \sigma \prec \pi_q. \quad (3.27)$$

Following the scheduling from $c_\emptyset$ to c_1, we are left with the problem of scheduling the t'_is from c_1 to c_{q+1}. The number of distinct due times is now q (note that $t'_i = \emptyset$ for every i such that $d_i = c_1$). Relabel the start time c_1 as $c'_\emptyset$ and the due times $c_2,\ldots,c_{q+1}$ as $c'_1,\ldots,c'_q$. Define $b''(i,j)$, $B''(i,j)$, and $C''(i,j)$ to be the values obtained for b, B, and C when t'_i is used in place of t_i and c'_j is used in place of c_j. We immediately see that $B''(i,j) = B'(i,j+1)$, and $C''(i,j) = C'(i,j+1)$, for every i, j, $1 \leq i \leq r$, $\emptyset \leq j \leq q$. Substituting into (3.27) yields:

$$\sum_{i=1}^{r} B''(i,\sigma(i)) \leq \sum_{i=1}^{r} C''(i,\sigma(i)) \text{ for every } \sigma \prec \pi_q.$$

It now follows from the induction hypothesis that the t'_is can be scheduled. []

From Theorem 1, it is clear that by repeatedly using COMPUTE_W to determine the amount to be scheduled in each interval, a feasible schedule can be obtained whenever such a schedule exists. Each time COMPUTE_W is used, we need to recompute b, a, Z, X, and Y. The time needed for this is O(nk)(note that recurrences 3.6 and 3.1∅ will be used to compute Z and X). The <u>for</u> loop may be executed in O(kr + n) time. We may assume that $r \leq n$ and so the complexity of COMPUTE_W is O(kn). The Kafura-Shen algorithm is of complexity $O(n)$. Hence, the overall computing time for the k phases of our scheduling algorithm is $O(k^2n)$. An additional $O(n\log n)$ time is needed to sort the jobs by memory size m_i. Hence, the overall complexity of our preemptive scheduling algorithm is $O(k^2n + n\log n)$.

4. Minimizing L_{max}

Let S be a preemptive schedule for (t_i,d_i,m_i), $1 \le i \le n$. Let f_i be the finish time of job i in S. If $f_i \le d_i$, $1 \le i \le n$ then S is a feasible schedule and no job is late. The lateness of job i is f_i-d_i and $L_{max} = \max\{f_i-d_i : 1\le i\le n\}$. Note that $L_{max} \le 0$ iff all jobs finish by their due times. Also, note that if $L_{max} \le 0$ then $c_0-c_1 \le L_{max}$.

From the definition of L_{max}, if follows that by changing the release time from c_0 to c_0-L_{max} we obtain a job set that can be scheduled such that no job finishes after its due time. Hence, to determine the minimum L_{max}, we need to determine the least x such that the condition of Theorem 1 is satisfied when a release time of c_0-x is used. This x may be obtained from a form equivalent to that of Theorem 1. We observe that $\sum_{i=1}^{r} B(i,\sigma(i)) \le \sum_{i=1}^{r} C(i,\sigma(i))$ for every $\sigma \prec \pi$ iff $\max_{\sigma\prec\pi}\{\sum_{i=1}^{r} B(i,\sigma(i)) - \sum_{i=1}^{r} C(i,\sigma(i))\} \le 0$. It is helpful to rewrite this form seperating out the case when $\sigma(i)=0$, $1 \le i \le r$. For this σ, we see that $\sum_{i=1}^{r} B(i,\sigma(i)) - \sum_{i=1}^{r} C(i,\sigma(i)) = \sum_{1}^{n} b(i,0)$. For every other σ, there is an s, $1 \le s \le r$ such that $\sigma(s) \ge 1$.

Define H_s as below:

$$H_s = \max_{\sigma\prec\pi,\sigma(s)\ge 1} \{\sum_{i=1}^{s} B(i,\sigma(i)) - \sum_{i=1}^{s} C(i,\sigma(i))\},\ 1\le s\le r.$$

We immediately see that

$$\max_{\sigma \prec \pi}\{\sum_{i=1}^{r} B(i,\sigma(i)) - \sum_{i=1}^{r} C(i,\sigma(i))\}$$

$$= \max\{\sum_{1}^{n} b(i,\emptyset),\ H_1,\ H_2,\ldots,\ H_r\}$$

Let $x_1 = \max_{1 \le i \le n} \{t_i - d_i + c_\emptyset\}$ and let $x_2 = \max_{1 \le i \le r} \{H_i/M_i\}$.

Clearly, if we change the release time to $c_\emptyset - \max\{x_1,x_2\}$ then $\max\{\ \sum b'(i,\emptyset),\ H'_1,\ H'_2,\ \cdots,\ H'_r\} = \emptyset$ (the b', H'_i values are computed with respect to the new release time $c_\emptyset - \max\{x_1,x_2\}$). Hence, $\max_{\sigma \prec \pi}\{\sum B'(i,\sigma(i)) - \sum C'(i,\sigma(i))\} \le \emptyset$ and $\sum B'(i,\sigma(i)) \le \sum C'(i,\sigma(i))$ for every $\sigma \prec \pi$. Moreover, $x = \max\{x_1,\ x_2\}$ is the least value of x for which this happens. Hence,

$$(L_{max})_{min} = \max\{x_1,x_2\}.$$

The H_ss may be computed in O(kn) time as follows. Define H_s^i as below:

$$H_s^i = \max_{\substack{\sigma \prec \pi \\ \sigma(s) \ge i}} \{\sum_{j=1}^{s} B(j,\sigma(j)) - \sum_{j=1}^{s} C(j,\sigma(j))\},\ 1 \le s \le r,\ 1 \le i \le k.$$

Hence, $H_s = H_s^1$, $1 \le s \le r$. We immediately obtain the following recurrence for H_s^i:

$$H_s^i = \begin{cases} \max\limits_{j \ge i}\{B(1,j) - C(1,j)\} & \text{if } s = 1 \\ \sum\limits_{j=1}^{s} B(j,k) - \sum\limits_{j=1}^{s} C(j,k) & \text{if } i = k \\ \max\{H_{s-1}^i + B(s,i) - C(s,i),\ H_s^{i+1}\} & \text{otherwise.} \end{cases}$$

Using this recurrence, all the H_s^i s may be obtained in O(rk) time (excluding the time needed to determine the b(i,j)s, B(i,j)s etc.). The additional time needed to compute the B(i,j)s and C(i,j)s is O(kn + nlogn) (assuming $n \geq m$). Hence, the minimum L_{max} may be determined in O(kn + nlogn) time. Having determined the minimum L_{max}, a schedule having this L_{max} value can be obtained by chaning $c_\emptyset$ to $c_\emptyset-(L_{max})_{min}$ and using the algorithm of Section 3.

5. Conclusions

We have developed an $O(k^2n + n\log n)$ algorithm to obtain a preemptive schedule for n jobs (t_i,d_i,m_i), $1 \leq i \leq n$ on m processors with given memory sizes. This schedule minimizes L_{max}. The minimum value of L_{max} can itself be obtained in only O(kn + nlogn) time.

References

1. Z. Galil and A. Naamad, "Network flow and generalized path compression," Proc. ACM Symp. Theo. of Comput., 1979, pp. 13-26.

2. M. Garey and D. Johnson, "Computers and intractability, a guide to the theory of NP-Completenss," W. H. Freeman and Co., San Francisco, 1979.

3. D. Kafura and V. Shen, "Task scheduling on a multiprocessor system with independent memories," SICOMP, Vol. 6, No. 1, 1977, pp. 167-187.

4. R. McNaughton, "Scheduling with deadlines and loss functions," Manag-Sci, 12, 7, 1959.

5. S. Sahni, "Preemptive scheduling with due dates," Op. Res., Vol. 27, No. 5, 1979, pp. 925-934.

6. S. Sahni and Y. Cho, "Scheduling independent tasks on a uniform processor systems," JACM, Vol. 27, No. 3, 1980, pp. 550-563.

7. S. Sahni and Y. Cho, " Nearly on line scheduling of a uniform processor system with release times," SICOMP, Vol. 8, No. 2, 1979, p275-285.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A103	3. RECIPIENT'S CATALOG NUMBER 795
4. TITLE (and Subtitle) Preemptive Scheduling Of A Multiprocessor System With Memories To Minimize L_{max}		5. TYPE OF REPORT & PERIOD COVERED Technical Report June 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ten Hwang Lai and Sartaj Sahni		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0650
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Minnesota 136 Lind Hall, 207 Church St. SE, Mpls.,MN 55455		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington VA 22217		12. REPORT DATE June 1981
		13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Preemptive scheduling, L_{max}, memory requirements.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We develop an $O(k^2n + n\log n)$ algorithm to obtain a preemptive schedule that minimizes L_{max} when n jobs with given memory requirements are to be scheduled on m processors ($n \geq m$) of given memory sizes. k is the number of distinct due dates. The value of the minimum L_{max} can itself be found in $O(kn + n\log n)$ time.

DATE
ILME